Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005

EVALUATION OF A SMALL SCALE CLUSTER COMPUTING SYSTEM FOR PARALLEL INTELLIGENT TECHNIQUES APPLICATIONS

CHUN-CHE FUNG[1], JIA-BIN LI[1], DOUGLAS MYERS[2]

[1]Centre for Enterprise Collaboration in Innovative Systems, School of IT, Murdoch University, Western Australia
[2]Department of Electrical and Computer Engineering, Curtin University of Technology, Western Australia
E-MAIL: L.Fung, J.Li@murdoch.edu.au, myersd@bauhaus.ece.curtin.edu.au

Abstract:

A small scale distributed computing system that is able to meet the needs of parallel intelligent techniques for engineering and science applications is reported in this paper. The reported system is a cluster of general-purpose PCs interconnected in a network. Such systems are powerful yet low cost. While such systems are not new, most applications written for cluster systems are programmed in MPI-C. Results from the proposed system have shown that it is advantageous to use Java due to its simplicity and mobility. Hence, the goal of this research is to design a Java-based parallel platform for efficient execution of intelligent techniques for engineering and science applications on a cluster system.

Keywords:

Cluster computing systems; computational intelligence; java; parallel and distributed programming

1. Introduction

Applications such as bioinformatics, climate prediction, biological analysis, chemical reaction analysis and power grid analysis are computational intensive and generally require a high performance computer system to obtain accurate analysis result within reasonable short time. In addition, the implementation of implicit parallel intelligent techniques such as neural networks and genetic algorithms places extra burden on the computation system. Traditionally, these highly computation demanded applications were executed on a conventional super computer such as vector computer or Massively Parallel Processors (MPP) systems, which normally would have cost millions of dollars to build. Further, these "big-box" machines are hard to program and inconvenient to maintain. Recent advance in network technologies has led to major improvement in the performance of cluster computing and grid computers. A cluster computing system consists of a group of commodity computers interconnected with some form of standard or special networks. Such cluster computing system installs a copy of operating system, normally Unix-based OS, on each node in the system and the system is capable to provide parallel computation for applications through a communication middleware. The current industry standard is by message passing interface (MPI). Grid computers are similar to cluster computers excepted that they normally do not provide parallel computation service. Rather, grid computers "steal CPU cycles" to provide backup computation services. A program running on a grid computer is very coarsely grained. That is, a sub-process does not need to collaborate with other sub-processes to obtain the result. Furthermore, time to complete the application is usually not the key constraint. However, it does not mean grid computers can not provide parallel computation as they are based on the same technologies used in cluster systems.

A cluster computing system generally includes six hierarchical layers as shown in Figure 1. The six layers are: internetworking, computation node, operating systems, compilers, distributed programming models, middleware and application. Different choice of technologies and configuration in each layer may vary the performance of the system. In addition, a cluster computing system is normally named based on technologies used to build the system. For instance, Beowulf cluster computing is constructed upon commodity PCs with Linux operating systems. System X, on the other hand, implemented 1100 Apple XServer G5 dual processor cluster nodes, where each node is running Mac OS X. There is no common rule to build a cluster computing system and so each cluster computing system is virtually different to others with respect to the technologies used and configurations of the system. That is, a cluster computing system is built accordingly to its requirements and specifications. Once a cluster system is built, the system will then be tuned to deliver its peek performance according to a selected range of applications.

0-7803-9091-1/05/$20.00 ©2005 IEEE

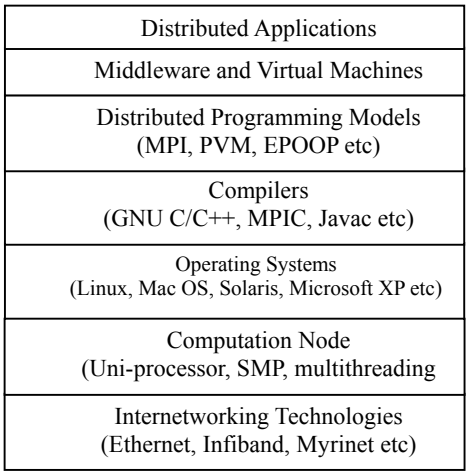

Figure 1: Hierarchy in cluster computing.

2. A Low cost personal cluster computing system – perCluster

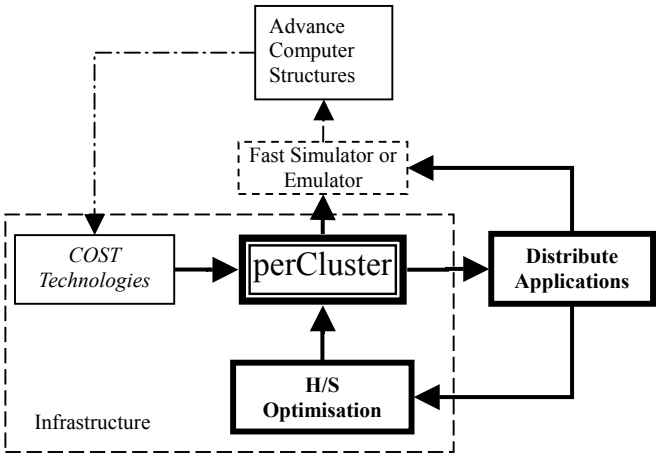

Figure 2: Overview of the perCluster project.

Figure 2 gives the overview of the perCluster project. As mentioned previously, the perCluster system is based on commodity technologies. Using the system as operating environment, a number of distributed applications will be developed. These distributed applications are categorized into the following groups.

- Computational Intelligent (CI) applications, including generic algorithms (GA), artificial neural networks (ANN) and fuzzy logic (FL); and
- Imaging processing, e.g. a content-based retrieval system.

These applications are developed based on a set of programming libraries that implement various parallel and distributed programming models. For instance, a client-server approach is a commonly used programming model in parallel machines. A client, or a worker, receives tasks scheduled by the server node and works till completion. There is, generally, no collaboration between different clients. The major advantage of such model is its simplicity. There is, however, a variant model where the scheduling process is not centrally managed. In the decentralized approach, upon completion of the previous task, a worker nominates itself to the next most appropriate task which matches with the specifications of the worker. To subscribe the task, the worker searches the list of tasks in the job database published by the server.

These distributed applications are used to benchmark the performance of the distributed programming libraries and the underlying hardware structures. In turn, the programming libraries and the hardware structures will be tuned to improve the performance of those applications. Optimization techniques will be developed to improve the solutions for real world problems.

In summary, the perCluster project focuses on the following areas:

1. Development of a low cost yet efficient cluster computing systems;
2. Development of distribute and parallel programming models;
3. Investigation into improved optimization techniques; and
4. To apply the system to solve real world problems.

3. Configuration of the perCluster system

Initially, perCluster includes one head node and nine computing nodes. All nodes are equipped with an uniprocessor running at 800MHz or 433MHz, a memory module (256MB – head node or 130MB – clients), a hard disk, a LAN adaptor and Redhat Linux 8.0 operating system. Ethernet is the only network used in perCluster due to its simplicity and low cost. A central 100 Mb/s Cisco switch interconnects all nodes to provide computation communication, file distribution, and system control.

4. Internetworking in perCluster

The current perCluster system uses a single 100Mbps based Ethernet network due to the cost concern. However, the next generation of the perCluster system will use giga-bit networks such as Gigabit Ethernet and Infiband when the price of giga-bit networks drops in the future. It is expected that the performance of the perCluster will scale up when faster networks are deployed. Because

communication is the most critical component in a cluster computing system, it is important to understand the characteristics of the network implemented in the system. To benchmark the network, we used the system command *ping*. The command sends a package to the destination and the package is returned back to the source. The ping command was used to simulate the point-to-point communication, which it is the basic communication pattern between two nodes. Various sizes of packages sent to obtain different round trip times were experimented. The same testing process was run on ten cluster nodes and the same package was sent three times to get the minimum, average and maximum Round Trip Time (RTT) for each node. Results were gathered and illustrated in Table 1. There are four columns defined in the table. The first column contains different sizes of package used in the testing. The second column shows the minimum round trip time spent in sending a particular message. Note that each value is an average value across ten cluster nodes. The third and fourth columns show the average RTT and maximum RTT in millisecond.

Table 1: The minimum, average and maximum round trip time (RTT) as function of the package size.

Package size (bytes)	Minimum RTT (ms)	Average RTT (ms)	Maximum RTT (ms)
64	0.15	0.16	0.17
128	0.17	0.18	0.19
256	0.21	0.22	0.24
512	0.30	0.31	0.32
1024	0.47	0.48	0.49
1472	0.63	0.65	0.68
2048	0.74	0.76	0.80
4096	1.10	1.11	1.13
8192	1.84	1.86	1.88
16384	3.32	3.34	3.37
32768	6.29	6.33	6.37
65536	12.31	12.37	12.44

Figure 3 depicts RTT as a function of the increasing package size. RTT is low (RTT < 1 msec) when the size of the message is below 2K bytes. The size of most packages used in computer networks is relatively shorter than those in data networks, because a message usually contains arguments such as integers or floating points for a remote process to compute. Hence it suggests that the perCluster system may perform well even using 100 Mbps Ethernet.

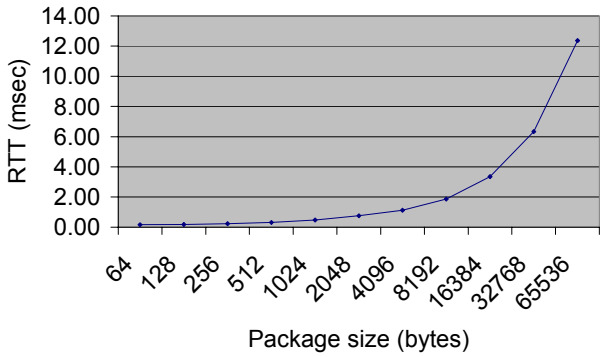

Figure 3: The average round trip time as function of the package size.

Figure 4 shows the effective data bandwidth as a function of the package size. The effective data bandwidth improved when the package size increased. When sending or receiving a message, an interrupt incurs to notify the operating system to handle the incoming or outgoing message. Such overhead results in degrading the CPU utilisation. Transmitting larger size packages will generally require less system interrupts. Consequently, the bandwidth of a network will be greater for transmitting larger size packages. Due to this reason, both jumbo frames that contain more payloads in a message and the combining and forwarding technique that merge multiple packages into one big package are used in cluster computing systems to improve communication performance. However, the effective bandwidth will become steady even the size of the package is very large. As shown in Figure 4, the effective data bandwidth saturated at around 10M Byte/sec ($\approx$ 80Mbps) where the package size was 64 Kbytes.

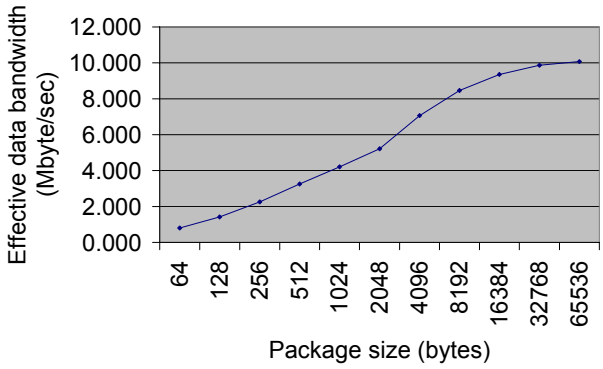

Figure 4: Effective bandwidth as function of package size sent.

5. Integrating perCluster with desktop applications

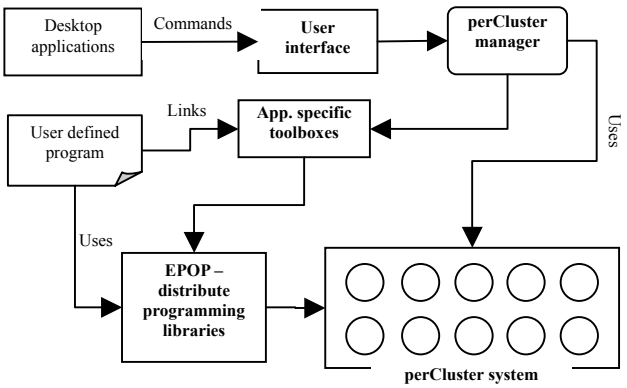

Figure 5: Seamless integration of the perCluster system to existing desktop applications via toolboxes and EPOP.

The perCluster system provides two ways for user interface. A user may interact with the perCluster system by typing commands to the perCluster manager. Through the user interface, a user can call any primitive functions predefined in the system. However, the user may want to use some domain specific libraries on solving some problems. A toolbox provides convenience to users to solve domain specific problems. As mentioned previously, two prime research areas in the perCluster group include computational intelligence and image processing.

5.1. Computational Intelligent (CI)

Computational Intelligent (CI) techniques primarily concern with three major disciplines: Artificial Neural Network (ANN), Fuzzy Logic (FL) and Evolutionary Computation (EC). Such techniques are frequently used to solve many real-life and complex problems in system engineering. Particularly, practical problems with non-linear and nondeterministic nature are often not applicable via traditional solution processes. Examples are many problems in electrical power system engineering which generally require massive computational efforts. Examples are the Economic Dispatch (ED) model to resolve the economic loadings of the electric generators so that the load demand can be met and the loadings are within the feasible operating regions of the generators [1]. Such process involves finding a global maximum point to satisfy the model. Fung et al [2] applied the evolutionary computation technique including implementation of Genetic Algorithm (GA), Simulated Annealing (SA), and Tabu Search (TS). Two implementations, namely, cluster structure and parallel structure, on a cluster system had been reported [2]. Results showed that the distributed implementations can improve the overall computational time to complete the problem. These models are available in the form of a CI toolbox.

5.2. Imaging Processing (IP)

Although it is possible to retrieve images from database using a unique identification defined by a human operator as an index to images, it is more convenient and natural to search images based on their contents. The principle of Content-Based Image Retrieval (CBIR) system is to retrieve images based on the content of the images. One of the important components in CBIR system is to extract the visual features of the images for performing more abstract analysis. However, deriving these features are computationally expensive. To solve this issue, a more flexible architecture is desired to improve the extraction time for the system. Consequently, a parallel framework based on the perCluster system has also been proposed aims at improving the feature analysis process [3].

6. Distributed programming libraries

Proposal of different distributed programming models had been reported [4]. The proposed programming models will be included in a distributed programming software package, termed Efficient Parallel Object-Oriented Platform (EPOOP). Similar to MPI [5] or PVM [6], EPOOP is able to coordinate a network of computers to be used as a single machine. However, EPOOP also addresses other types of computational models such as distributed shared memory (DSM) model and Actor model. There are three general design approaches to implement parallel computers: Multiprocessors, Multi-computers, and Distributed Shared Memory (DSM). In the case of a multi-processor design, all the processors are confined and share the resources within a single, global address space. Read and write operations are both allowed for any processor to access the common address space. Communication among processors is thereby made through the shared memory. This architectural design is also termed as a tightly coupled parallel system. On the other hand, a multi-computers system comprises of a network of computers. Every computer in the network has its own private memory space and I/O modules. Hence communication is via message-passing. For example, the MPI standard is commonly accepted by the industry.

In practice, multiprocessors systems are costly and complicated to build but they are relatively easy to program. On the other hand, multi-computers are just the opposite. Distributed Shared Memory architecture can be considered as an intermediate solution which takes advantages of the

two design philosophies. In DSM, each computer has both local and shared memory spaces. The shared memory space is visible to all remote processors located in the same network. From a programmer's perspective, the read or write operations on the shared or local variables are being treated the same irrespective to their physical locations. In other words, variables stored in the shared memory space can be accessed by a remote processor in a same manner as variables being stored in the processor's local memory space.

Similar to cache coherence problem in multiprocessor design, the shared memory locations of a DSM system must be protected and synchronized to ensure data consistency. There have been a number of consistent protocols for the DSM design proposed in the past years. Examples of such developments are Sequential Consistency [7], Lazy Release Consistency [8] and Entry Consistency. EPOOP will improve the performance of those consistency models by allowing adaptive consistency models. That is, EPOOP is able to automatically choose between the different protocols depending on the access patterns of the application and the workload in the network. It is recognized that the performance of a protocol is application dependent. In contrast to the original Para Worker, the proposed system allows dynamically allocate processes across different computers to balance the workload on the system. Actor model is another concurrent computational model proposed by Agha [9]. The basic concept in the actor model is that every object is active and the object communicates with others by passing messages. An actor is quite similar to a thread object in Java. However, actors are message-driven while threads tend to focus on a shared memory model.

7. CPI: Evaluation of perCluster performance

In this section, the performance of the perCluster system is evaluated with a benchmark program called CPI. CPI is an example program included in the MPICH software – a portable implementation of MPI [10]. CPI simply computes the π value by using integral approximation (see Eq. 1). Basically, the larger number of N will provide a more accurate π value.

$$\pi = \int_{a=1}^{N} \frac{4}{1 + (\frac{a - 0.5}{N})^2} \qquad (1)$$

For the large number N, it is obvious that the program will run a great deal of loops to compute the π value. However, Matlab performs badly for loop-intensive programs. Figure 6 depicts the computation time of a program as a function of loop counts. The relationship between CPU time and the loop number is nearly linear with the gradient of 0.00173. That is, to complete a 105 loops task requires the computer to run approximately 173 sec.

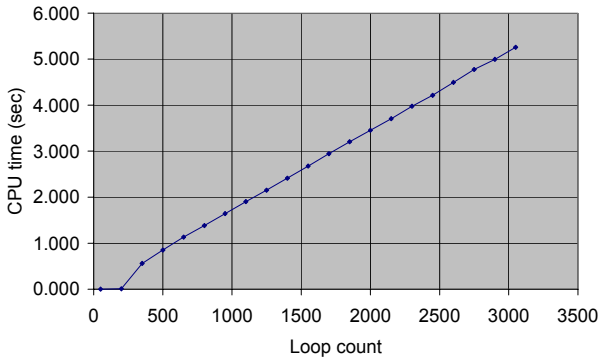

Figure 6: Illustration of execution time as function of loop distance in Matlab. In every loop, the program called a *FFT* function with the input of a 2-dimensional matrix.

In terms of the CPU time spent on running the Matlab version CPI program as a function of N (accuracy), the computation time was less than 2 sec for N is less than 105. However, when N is greater than 106, the total computation time increased dramatically. For N=107, Matlab ran nearly 250 sec to complete the CPI program.

The MPI implementation for CPI was subsequently created in Java, termed JCPI, running on a server-client model. A server is responsible to distribute the initial parameters and to collect results from the clients. A client, also known as worker, computes only a subset of the workload, which it is the sub-range of the loops in such case, and it submits its result to the server after completion. There is no collaboration needed for clients. Hence each runs independently during execution. Figure 7 illustrates a 3D view of the benchmarking result running JCPI on the perCluster system. The x-axis represents the workload or accuracy; y-axis represents different sizes of nodes; and z-axis represents the CPU time in the unit of second. For every column on the y-axis, the computation time increases as result of increasing workload. However, the rate of change is not the same across different size of nodes. The computation time boosts from 0.1 sec to 0.6 sec at workload of 107 on y = 1, i.e. the size of the system is one, while the computation time lifts only 0.1 sec at the same workload when the size of the system becomes ten.

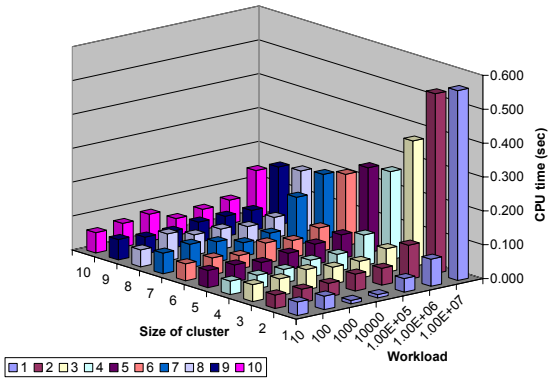

Figure 7: CPU time as function of size of cluster and workloads with the Java version of CPI.

To demonstrate the performance gained from the perCluster, we computed the speed-up of JCPI against the Matlab version CPI. Speed-up can be obtained using the following formula:

$$\text{speed} - \text{up} = \frac{\text{T(series)}}{\text{T(parallel)}} \qquad (2)$$

It is obvious that there is performance gained from a parallel system if the speed-up is greater than unity or one. Different sizes of the system, 2, 4, 6, 8, and 10 have been tested. In general, speed-up of the perCluster is not over unity till the workload is larger than 1000. When N is 106, the speed-up of the system with the size of 2 is 245. At the same workload the speed-up of the system with the size of 10 is approximately 354. That is the larger size of the system performed better in such application. Considering the speed-up of the system using MPI-C implementation, the performance improved as the size of the system increased. However, the system gets significant speed-up ($S \approx 22$) comparing with Matlab even for the small amount of workload (N=100). Although Java-based CPI has slow start-up, it is compatible with the performance of MPI-C at N=106. For N>106 the performance of the Java-based CPI is even superior to that of the MPI-C. It can therefore be concluded that it is not necessarily true that Java is always slower than C or C++.

8. Conclusions

This paper presents a Java-based distributed framework that utilizes cluster computing for the implementation of intelligent techniques for computation of complex engineering and science applications such as image processing. Performance of the system is illustrated by running a basic function that computes π values. The Java implementation had been tested against MPI/C and Matlab native functions. Results from experiments have shown that Java implementation is superior to Matlab native functions for the problem domain is greater than 1000. Furthermore, the result also shows that performance of Java implementation is better than MPI/C for a large size of nodes. It demonstrates the feasibility of the proposal and further development is continued.

Acknowledgements

This paper is supported by the Murdoch University Research Excellence Grant Scheme and Mr J.B. Li is supported by an International Postgraduate Research Scholarship.

References

[1] Wong K.P & Wong Y.W, "Genetic and genetic/simulated-annealing approaches to economic dispatch"; *IEEE Proceedings of Genetic Transmission Distribution* Vol. 141, No.5, September 1994, pp 507-513.

[2] Fung, C. C., Chow, S. Y. & Wong, K. P., "A Low-Cost Parallel Computing Platform for Power Engineering Applications", (APSCOM'00), *Proceedings of Advances in Power System Control, Operation and Management*, Hong Kong 2000, pp. 354-358.

[3] Chung K.P., Li J.B., Fung C.C. and Wong K.W., "A parallel architecture for feature extraction in content-based image retrieval system", *Proceedings of the 2004 IEEE International Conference on Cybernetics and Intelligent Systems (CIS 2004)*, Dec. 2004, Singapore, pp. 468-473.

[4] Fung C.C., Li J.B. and Wong K.W. (2004), "Development of a Java-based distributed platform for the implementation of computational intelligence techniques", *Proceedings of the Third International Conference on Machine Learning and Cybernetics (ICMLC)*, Aug. 2004, Vol. 7, Shanghai, pp. 4156-4161.

[5] The Message Passing Interface (MPI) standard, [URL] http://www-unix.mcs.anl.gov/mpi/ Accessed on 10/04/2005

[6] Parallel Virtual Machine (PVM), [URL] http://www.csm.ornl.gov/pvm/ Accessed on 10/04/2005

[7] K. Li and P. Hudak, (1989), "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactions on Computer Systems*, v 7, n 4, Nov. 1989, p 321-59.

[8] C. Amza, A. L. Cox, S. Dwarkadas, L.-J. Jin, K. Rajamani, and W. Zwaenepoel, (1999), "Adaptive protocols for software distributed shared memory" *Proceedings of the IEEE*, v 87, n 3, March 1999, p.p. 467-75.

[9] G. A. Agha (1986), "Actors: A Model of Concurrent Computation in Distributed Systems", Cambridge, Mass. MIT Press, 1986.

[10] MPICH – a portable implementation of MPI, [URL] http://www-unix.mcs.anl.gov/mpi/mpich/ Accessed on 10/04/2005